University of Petra

307307
Generative AI

Introduction to Transformers and Context Aware Embeddings

Introduction to Transformers

Transformers – Architecture and Principles

What is a Transformer?

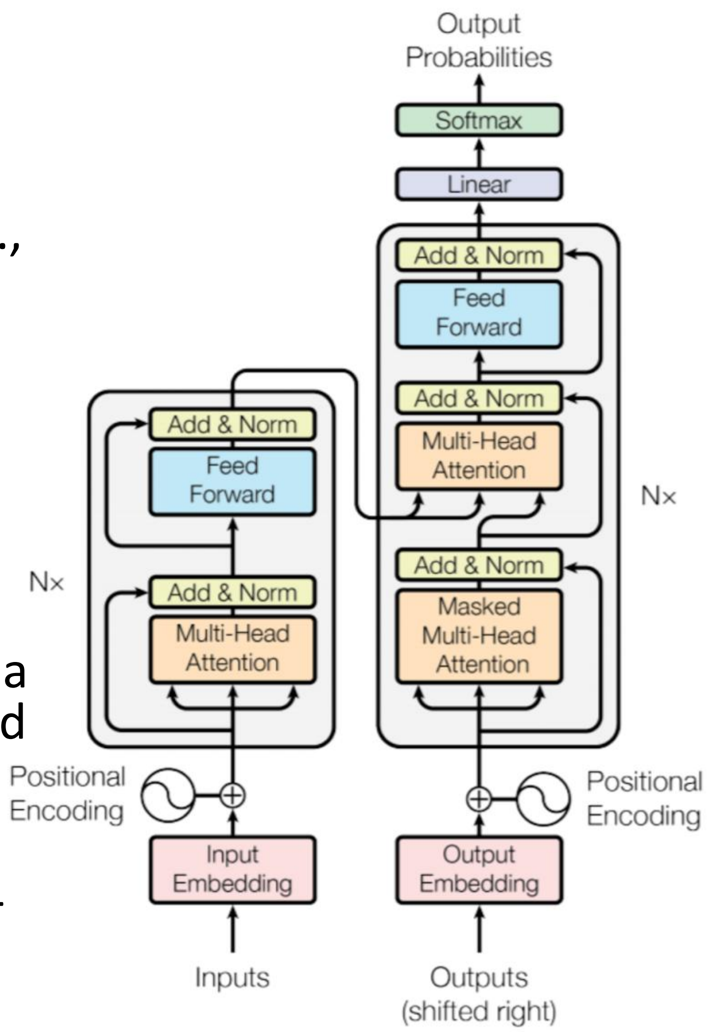
- A deep learning model based entirely on **self-attention**, with no recurrence or convolutions
- Introduced in the paper *"Attention Is All You Need"* (Vaswani et al., 2017)

The transformer model consists of two main parts:

1. Encoder: The encoder processes the input sequence and generates a continuous representation of it. This representation captures the contextual information of the input tokens.

2. Decoder: The decoder takes the encoder's output and generates the final output sequence. It does this by predicting one token at a time, using the encoded representations and previously generated tokens.

Both the encoder and decoder are composed of multiple identical layers—typically six layers in the original transformer architecture—allowing for deep learning and complex feature extraction.

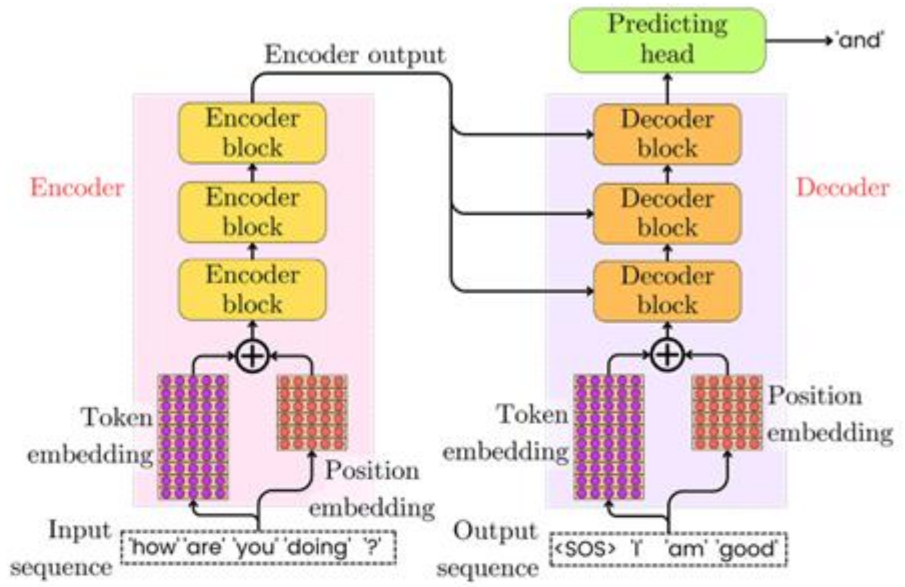


A Summary of how the Transformer Works

Input Sentence: "The cat chased the mouse.“

Input Encoding:
- Break down the sentence into tokens (words).
- Convert each token into a numerical representation called an embedding.
- Add positional encodings to the embeddings to provide information about the position of each token.

Encoder Processing:
- The encoder takes the input embeddings with positional encodings.
- Pass the input through multiple layers of multi-head attention and feed-forward networks.
- Each encoder layer processes the input, allowing the model to learn complex representations of the sentence.
- The attention mechanism in the encoder learns relationships between tokens (e.g., "cat" is related to "chased" and "mouse").



A Summary of how the Transformer Works

Decoder Processing:

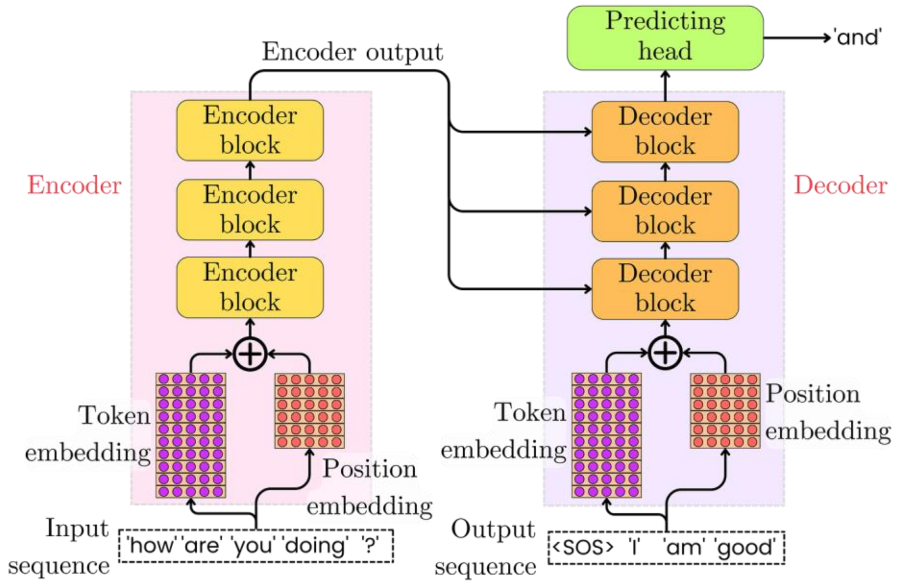
- The decoder takes the encoder's output.
- Use masked multi-head attention to generate the output one token at a time.
- Attend to the encoder's output to incorporate context from the input sentence while generating the translation.
- The attention mechanism in the decoder focuses on relevant parts of the input (e.g., the representation of "cat" when generating "Le chat").

Output Generation:

- The decoder generates the output sequence token by token.
- For the example, it generates: "Le", "chat", "a", "poursuivi", "la", and "souris".
- The complete French translation is: "Le chat a poursuivi la souris."

Key Advantages:

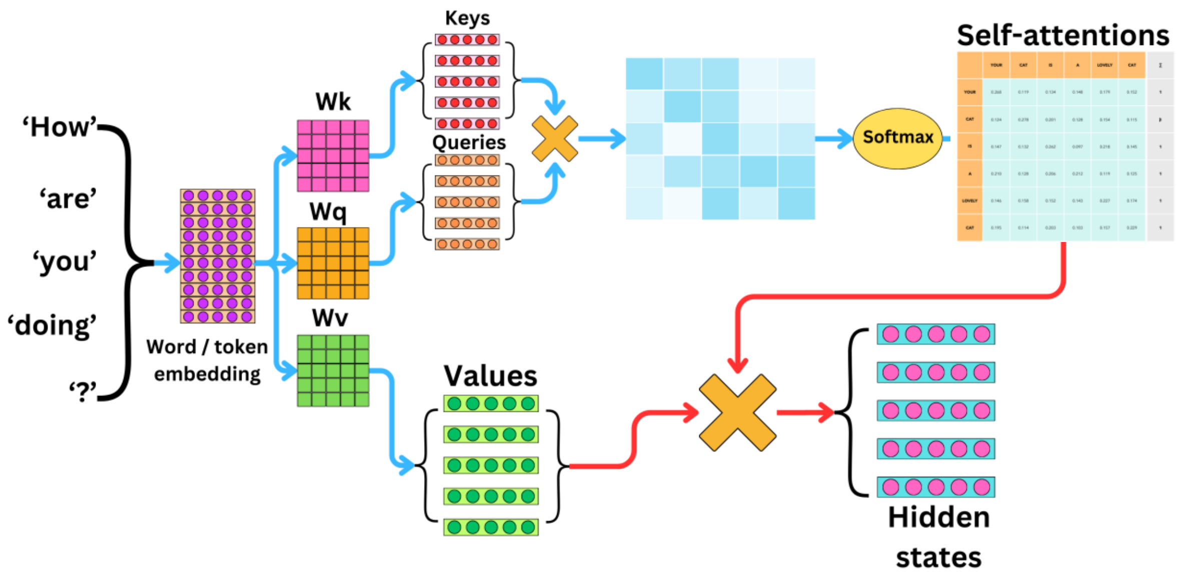
- Process the entire input sequence simultaneously.
- Use attention mechanisms to capture relationships between tokens.
- Efficiently translates sentences, even with long-range dependencies.



The Attention Mechanism

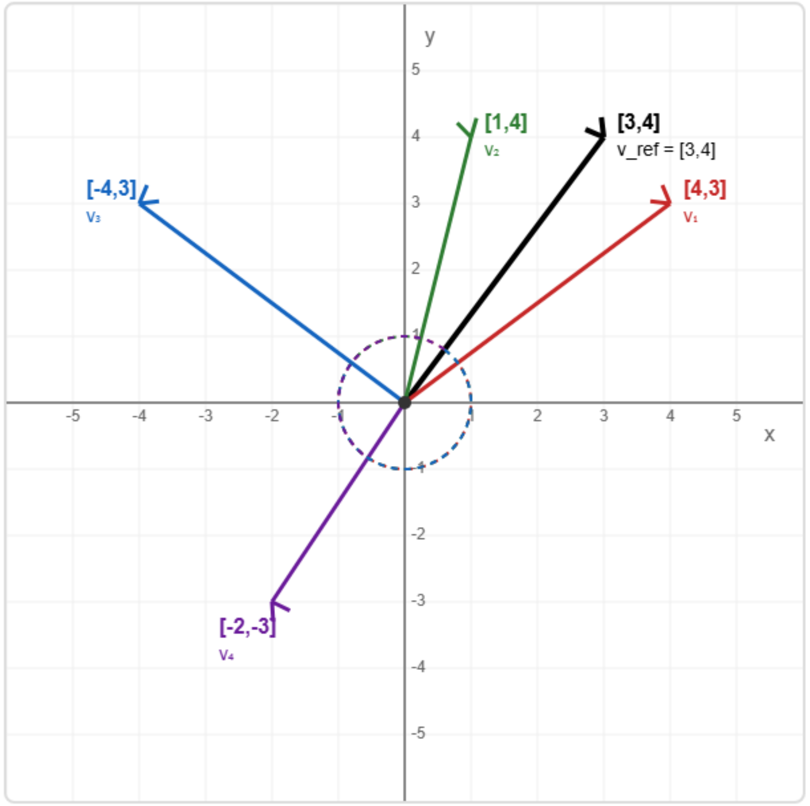
Self-Attention

- **Self-attention is how a transformer lets each word look at *other words in the same sentence* to understand context.**
- Each word asks: *"Which other words are important for understanding my meaning?"*
- Example:
  In the sentence **"The cat sat on the mat because it was tired."**
  The word **"it"** needs to look at **"cat"** to understand what "it" refers to.
  Self-attention makes this possible.
- **In short:**
  Self-attention helps a model understand relationships *within* the same sequence.



Vector Dot Product as Similarity Measure

$$v_1 \cdot v_2 = x_1x_2 + y_1y_2$$



1. Very Similar (High Positive)

`Red Vector v₁ = [4, 3]`

$v_1 \cdot$ v_ref = (4×3) + (3×4) = 24

Almost same direction as reference

2. Similar (Positive)

`Green Vector v₂ = [1, 4]`

$v_2 \cdot$ v_ref = (1×3) + (4×4) = 19

Generally same direction

3. Neutral (Zero)

`Blue Vector v₃ = [-4, 3]`

$v_3 \cdot$ v_ref = (-4×3) + (3×4) = 0

Perpendicular (90° angle)

4. Dissimilar (Negative)

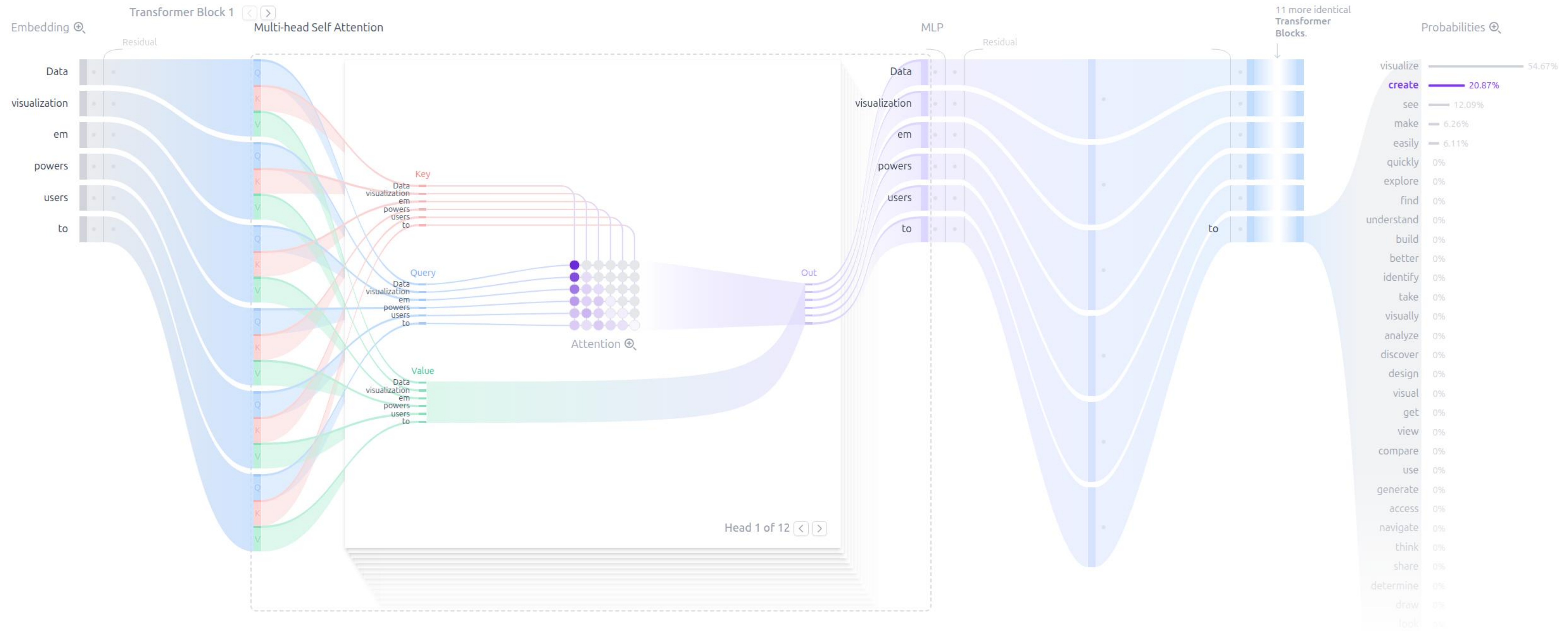
`Purple Vector v₄ = [-2, -3]`

$v_4 \cdot$ v_ref = (-2×3) + (-3×4) = -18

Opposite direction

Transformer Explainer

https://poloclub.github.io/transformer-explainer/



How LLMs Are Built

- **Step 1: Training an LLM (Pretraining)**
- Train a model from scratch on extremely large datasets.
- Learns general language structure, reasoning, and broad knowledge.
- Requires massive compute, long training cycles, and large-scale data cleaning.
- Produces a versatile, general-purpose base model.
- **Step 2: Fine-Tuning an LLM**
- Start with the pretrained model and adapt it to a specific task or domain.
- Uses a smaller, focused dataset.
- Much cheaper and faster than pretraining.
- Improves performance on targeted applications (support chatbot, coding assistant, legal summarizer, etc.).

Key Generation Settings in Language Models

Context Window

- The maximum amount of text the model can consider at once.
- A larger context window lets the model remember more of the conversation or document, improving coherence over long inputs.
- Once the limit is reached, older text is no longer considered.

Temperature

- Controls randomness in generation.
- Low temperature (e.g., 0.2) makes the model more focused and deterministic.
- High temperature (e.g., 1.0) makes the model more creative and varied.

Top-N / Top-K Sampling

- Limits the model to choosing from the top K most likely next tokens.
- Lower K makes output more predictable.
- Higher K adds more variety while still avoiding extremely unlikely tokens.
- If you want, I can format this as bullet points for a real slide deck or export it to PowerPoint.

Context Aware Embeddings

Contextual Word Embeddings (BERT and GPT)

Key Innovation

Unlike static embeddings Word2Vec, contextual models generate **different vectors for the same word** depending on its context.

Approach

- Uses deep, pre-trained neural networks (often transformer-based)
- Embeddings are derived from entire sentences, capturing syntax and semantics dynamically

Examples

- **BERT - Bidirectional Encoder Representations from Transformers (2018)**: Transformer-based neural networks trained with masked language modeling and next sentence prediction
- **GPT - Generative Pre-training Transformer (2018):** Transformer-based unidirectional language model focused on generation.

Characteristics

- Embeddings are **context-sensitive** (e.g., “bank” in “river bank” vs. “savings bank”)
- Each word is embedded based on its role in the sentence.
- Embeddings vary for the same word depending on its position and meaning.
- Significantly improve performance on downstream NLP tasks.

BERT: Bidirectional Encoder Representations from Transformers

- Developed by Google in 2018
- A **pre-trained language model** based on the **Transformer encoder**
- Reads text **bidirectionally**, enabling deep contextual understanding

Key Ideas

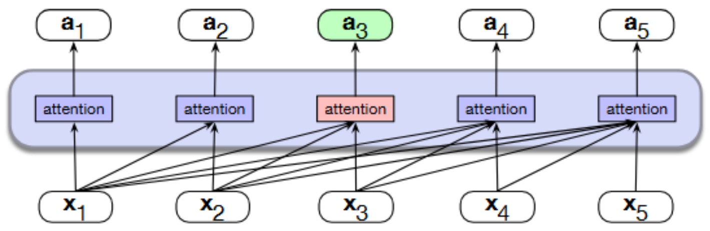
- Uses only the **encoder** stack of the Transformer
- Pre-trained on large text corpora, then fine-tuned on specific tasks

Pretraining Objectives

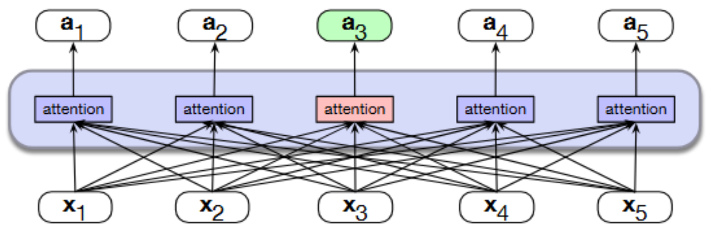
- **Masked Language Modeling (MLM)**: Predict randomly masked words in a sentence
- **Next Sentence Prediction (NSP)**: Predict if one sentence follows another

Applications

- Sentiment Analysis
- Question Answering
- Named Entity Recognition
- Text Classification
- Semantic Search


a_1 a_2 a_3 a_4 a_5
attention attention attention attention attention
x_1 x_2 x_3 x_4 x_5

a) A causal self-attention layer


a_1 a_2 a_3 a_4 a_5
attention attention attention attention attention
x_1 x_2 x_3 x_4 x_5

b) A bidirectional self-attention layer

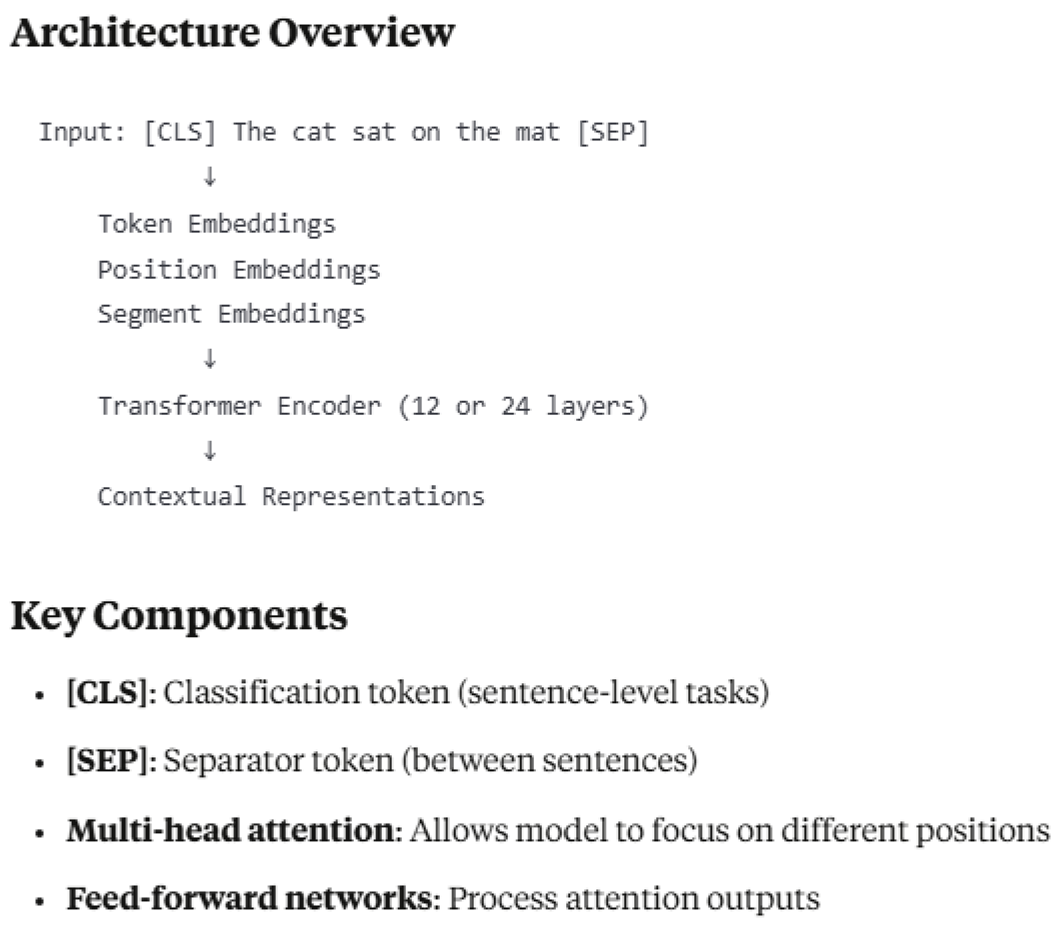
More About BERT

Key Advantages

- Contextual embeddings: Word meanings change based on context
- Captures long-range dependencies
- Pre-trained on massive datasets → Transfer learning
- State-of-the-art performance on 11 NLP tasks when released

How BERT Works:

- **Multiple stacked encoder layers** with self-attention mechanisms
- **No decoders** - purely focused on understanding input
- **Captures relationships** between words and their context
- **Powerful context learning** - understands word relationships in sentences
- **Meaningful representations** - transforms text into useful numbers



Architecture Overview

```
Input: [CLS] The cat sat on the mat [SEP]
           ↓
    Token Embeddings
    Position Embeddings
    Segment Embeddings
           ↓
    Transformer Encoder (12 or 24 layers)
           ↓
    Contextual Representations
```

Key Components

- **[CLS]**: Classification token (sentence-level tasks)
- **[SEP]**: Separator token (between sentences)
- **Multi-head attention**: Allows model to focus on different positions
- **Feed-forward networks**: Process attention outputs

BERT's Training Data

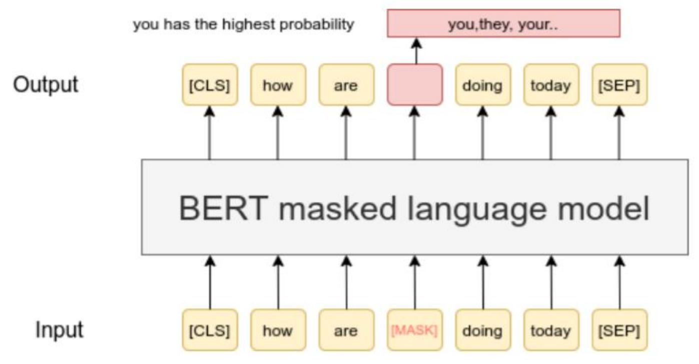
Pre-training Foundation

- **Training corpus**: 3+ billion words
  - English Wikipedia
  - ~10,000 unpublished books
- **Clean, large-scale dataset** for comprehensive language learning
- **Pre-training approach** to learn language patterns and context

Training Objective 1 - Masked Language Modeling

Learning Through "Fill in the Blanks"

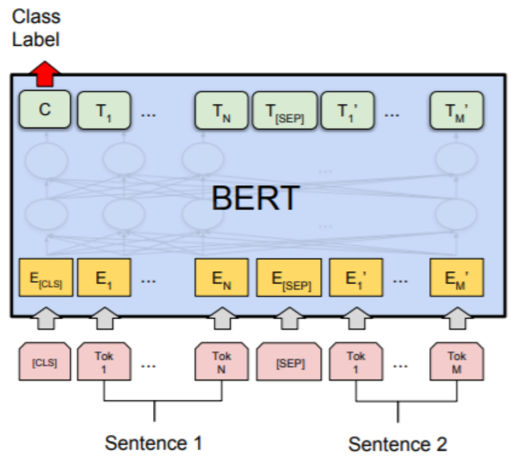
- **Random word masking**: Some words are hidden during training
- **Context-based prediction**: BERT predicts masked words using surrounding context
- **Example**: "I love eating _____ in my fruit salad" → "lemons"
- **Key benefit**: Learns word relationships and contextual understanding
- **Powerful concept**: Used in many other AI applications



BUSINESS INTELLIGENCE AND DATA ANALYTICS

Training Objective 2 - Next Sentence Prediction

- **Title: Understanding Sentence Relationships**
- **Sentence pair analysis**: Given two sentences, determine if B follows A
- **Logical connection assessment**: Analyzes context and content
- **Example**:
  - A: "The cat climbed the tree"
  - B: "It was trying to catch a bird" ✓ (follows logically)
  - C: "The weather is nice today" ✗ (unrelated)
- **Note**: Later removed in newer models (MLM proved sufficient)



(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG

BERT's Ideal Use Cases

Where BERT Excels

- **Text Classification**: Sentiment analysis, topic classification, spam detection
- **Named Entity Recognition**: Identifying people, organizations, locations, dates
- **Extractive Question Answering**: Finding answers within provided context
- **Semantic Similarity**: Measuring similarity between sentences/paragraphs
- **Key advantage**: Perfect for understanding tasks, not generation

Extractive Question Answering Example

How BERT Answers Questions:

- **Process**: Analyzes both context and question
- **Method**: Identifies start and end tokens of the answer
- **Example**:
  - Context: "Mount Everest is the highest mountain..."
  - Question: "What is the highest mountain?"
  - Answer: "Mount Everest" (extracted, not generated)
- **Important**: BERT extracts existing text, doesn't generate new content

Example: Print out BERT Embeddings

```
from transformers import BertTokenizer, BertModel
import torch

# Load pretrained BERT
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
model = BertModel.from_pretrained('bert-base-uncased’)

# Sentence
sentence = "He went to the bank to deposit money.“

# Tokenize
inputs = tokenizer(sentence, return_tensors='pt’)

# Get outputs
with torch.no_grad():  # No training, just inference
    outputs = model(**inputs)

# Get the hidden states (embeddings)
embeddings = outputs.last_hidden_state  # Shape: (batch_size, sequence_length,

# (hidden_size)
print(embeddings.shape)  # Example output: torch.Size([1, 11, 768])
```

Open in Colab

GPT – Overview and Architecture

What is Generative **P**re-trained **T**ransformer (GPT)?

- A family of **Transformer-based language models** developed by OpenAI
- Uses only the **decoder stack** of the original Transformer architecture
- Trained with **causal (autoregressive) language modeling** to predict the next token
- Focuses on generating human-like text (unlike BERT's understanding focus)

Training Objective

- Predict the next token in a sequence

GPT Variants

- **GPT-1 (2018, 120M Parameters)**: Introduced the pretrain-then-finetune paradigm
- **GPT-2 (2019, 1.5B Parameters)**: Scaled up model size, trained on web-scale data, Last publicly available weights
- **GPT-3 (2020, 175B Parameters)**: Enabled in-context learning, 100x boost, ~800GB file size
- **GPT-4 (2023, ~1T parameters (estimated))**: Multimodal, stronger reasoning and generalization

Applications (Wide range of NLP tasks through text generation)

- Text generation (e.g., chat, storytelling, code)
- Summarization
- Translation
- Question answering
- Semantic search and reasoning tasks

GPT's Architecture Deep Dive

How GPT Works

- **Multiple stacked decoder layers** with self-attention mechanisms
- **No encoders** - purely focused on text generation
- **Unidirectional processing** - considers only left-side context
- **Next word prediction** based on preceding words
- **High-quality text generation** from learned context patterns
- **Contextual understanding** combined with text creation capability

GPT Training Data

Pre-training Foundation

- **Massive, diverse datasets** including:
  - Web pages
  - Books and articles
  - Billions of words total
- **Different datasets** for each GPT version
- **Large-scale exposure** to language patterns and context
- **Quality varies** but emphasis on diversity and scale

Causal Language Modeling

GPT's Single Training Objective

- **Core task**: Predict the next word in a sequence
- **Method**: Uses context from preceding words only
- **Example**: "I love drinking fresh ___" → "lemonade"
- **Key benefits**:
    - Learns word relationships and context
    - Develops language patterns
    - Enables coherent text generation
- **Simplicity**: No masked language modeling or next sentence prediction

GPT's Ideal Use Cases

Where GPT Excels

- **Text Generation**: Story creation, creative writing, conversations
- **Instruction Following**: Responding to prompts and commands
- **Translation**: Converting text between languages
- **Summarization**: Creating concise summaries of longer texts
- **Code Generation**: Programming assistance and code completion
- **Versatile Applications**: Any task involving text output

GPT vs BERT Architecture

Aspect	GPT	BERT
Architecture	Decoder-only	Encoder-only
Text Processing	Unidirectional (left-to-right)	Bidirectional
Primary Goal	Text generation	Text understanding
Context	Previous words only	All surrounding words
Use Cases	Generation tasks	Classification/extraction

What is Hugging Face? 🤔

- **A company and a community platform** focused on democratizing Artificial Intelligence, especially Natural Language Processing (NLP) and Machine Learning (ML).
- Often called the "**GitHub for Machine Learning**."
- **Mission**: To make state-of-the-art ML models, datasets, and tools accessible to everyone.
- Started in 2016, initially with a chatbot app, then pivoted to open-source ML.

What does hugging face provide?

1. **Accessibility**: Provides easy access to thousands of pre-trained LLMs.
2. **Standardization**: Offers standardized tools and interfaces for working with different models.
3. **Collaboration**: Fosters a vibrant community for sharing models, datasets, and knowledge.
4. **Innovation**: Accelerates research and development in the LLM field.
5. **Ease of Use**: Simplifies complex ML workflows, from data preparation to model deployment.

Core Components of the Hugging Face Ecosystem

- **Hugging Face Hub**:
  - The central place to find, share, and collaborate on models, datasets, and ML applications (Spaces).
  - Over 1.7 million models, 75,000+ datasets!
- **Transformers Library**:
  - Python library providing thousands of pre-trained models for NLP, Computer Vision, Audio, and more.
  - Supports PyTorch, TensorFlow, and JAX.
  - Makes downloading, training, and using state-of-the-art models incredibly simple.
- **Datasets Library**:
  - Efficiently load and process large datasets.
  - Optimized for speed and memory, built on Apache Arrow.
  - Access to a vast collection of public datasets.
- **Tokenizers Library**:
  - Provides high-performance tokenizers crucial for preparing text data for LLMs.
  - Offers various tokenization algorithms and pre-trained tokenizers.

The Model Hub

Over 1,700,000+ Models Available

Popular Model Categories:

- **Text Generation**: GPT, LLaMA, Mistral, CodeLlama
- **Text Classification**: BERT, RoBERTa, DeBERTa
- **Question Answering**: BERT-based models
- **Translation**: T5, mT5, NLLB
- **Code Generation**: CodeT5, StarCoder
- **Multimodal**: CLIP, BLIP, LLaVA

Getting Started with Hugging Face

- Explore the Hub: huggingface.co
- Browse models, datasets, and Spaces.
- Install Libraries:

```
pip install transformers datasets tokenizers
accelerate gradio
```

- Try a Pipeline:

```
# Example: Sentiment Analysis
from transformers import pipeline
classifier = pipeline("sentiment-analysis")
result = classifier("Hugging Face is awesome!")
print(result)
# Example: Text Generation
generator = pipeline("text-generation")
output = generator("In a world of large language models,",
   max_length=50)
print(output)
```

Under the Hood

1. **Automatic model selection**
2. **Tokenization handled**
3. **Inference optimization**
4. **Result formatting**
5. **Device management**

Traditional Approach

1. Load tokenizer

2. Preprocess text

3. Load model

4. Run inference

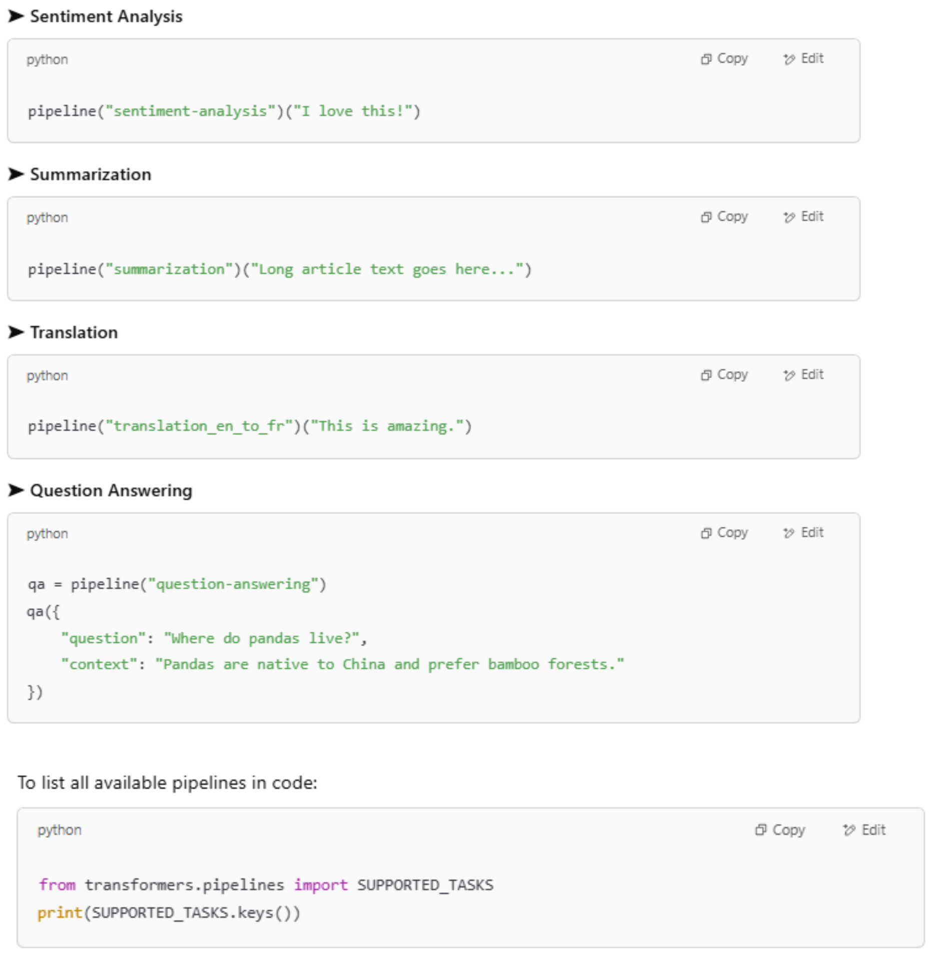
5. Post-process results

... 50+ lines of code

Other Hugging face Pipelines

The Hugging Face `transformers` library supports a wide range of **pipelines**, each designed for a specific **natural language processing (NLP)** or **vision** task — so you can use powerful models without deep setup.

Pipeline Name	Task Description
"sentiment-analysis"	Classify sentiment (positive/negative)
"text-classification"	General text classification (multi-label or multi-class)
"zero-shot-classification"	Classify into labels **without training** on them
"text-generation"	Generate text (e.g., GPT models)
"text2text-generation"	Text-to-text tasks (e.g., summarization, translation)
"translation"	Translate between languages
"summarization"	Generate a summary of input text
"question-answering"	Extract answer from context
"fill-mask"	Predict missing word in a sentence (BERT-style)
"ner" (Named Entity Recognition)	Extract entities (like names, places, etc.)
"conversational"	Chatbot-style conversation
"sentence-similarity"	Measure similarity between two sentences
"token-classification"	Classify each token (used for NER, POS tagging, etc.)
"feature-extraction"	Extract embeddings/features from a model
"table-question-answering"	QA over structured data (tables)



➤ **Sentiment Analysis**

```python
pipeline("sentiment-analysis")("I love this!")
```

➤ **Summarization**

```python
pipeline("summarization")("Long article text goes here...")
```

➤ **Translation**

```python
pipeline("translation_en_to_fr")("This is amazing.")
```

➤ **Question Answering**

```python
qa = pipeline("question-answering")
qa({
    "question": "Where do pandas live?",
    "context": "Pandas are native to China and prefer bamboo forests."
})
```

To list all available pipelines in code:

```python
from transformers.pipelines import SUPPORTED_TASKS
print(SUPPORTED_TASKS.keys())
```

Open in Colab